

METHOD AND SYSTEM FOR MANAGING DATABASE SQL STATEMENTS
IN WEB BASED AND CLIENT/SERVER APPLICATIONS

CROSS-REFERENCE TO RELATED APPLICATION

5 This application claims the benefit of Provisional Patent Application
No. 60/452,643, filed March 6, 2003.

TECHNICAL FIELD

10 The present invention is related to application development and
embedded relational database calls within application programs, and, in particular, to
a method and system for providing database-independent and embedded-SQL/stored-
procedure-independent calls for application development that can be easily toggled to
function as interfaces to various different DBMSs and to function either as embedded
database calls or as stored procedure calls.

15

BACKGROUND OF THE INVENTION

20 Typical problems associated with developing applications that
interface to relational DBMSs include: the disconnect between a fluid design-time
application environment vs. the "locked-down" run-time application environment;
applications that break when database or systems architecture's change; and non-
modular content and form components that have intertwined SQL statements and
application code.

25 All of these problems are exacerbated by the increased use of SQL as
a point for integration between applications, and as a source of business object rules.
Applications continue to migrate towards database centric solutions, where more
logic is stored with the database content rather than within the application code.
Unfortunately, many developers embed much of the SQL used to access these
business data and rules into the application code itself, negating some of the benefits.

One approach is to make a DBMS Stored Procedure call from the application code rather than building an SQL statement in the application code then calling the DBMS engine. But working with Stored Procedures is more cumbersome than file based SQL especially with other third party tools like source control, report
5 writers, template design tools and merge utilities. Even SQL Query development tools delivered with the DBMS software or provided by third parties work more simply with file based SQL statements rather than stored procedures. Ultimately however, SQL wants to run as a stored procedure within the database for maximum speed and efficiency. This is the “disconnect” between design-time and run-time
10 SQL development. Development tools are geared around building SQL statements in a file based architecture, and production databases are built around SQL statements coded within the database server.

In some cases application code is modified, typically at the last minute, to change from embedded SQL to embedded SP calls. But this means
15 changing the code back to embedded SQL in order to easily add major features or it means modifying the stored procedure directly to correct SQL mistakes. When SQL code is done in stored procedure it becomes much more difficult to port to other database platforms as SP syntax and “language” definitions vary more greatly between DBMS then does the SQL syntax, normally written in the SQL-92 standard.
20 Even many SQL statement syntaxes are DBMS particular and if these become embedded in the application code then the code is not portable between DBMS. One advantage to moving to stored procedures though is that applications have better separation between form or UI and application content making any application changes simpler. Either way there is no simple way to use the third party dev tools to
25 test, debug and correct the SQL statements.

So if you keep the code as embedded SQL you have better DBMS platform portability but more difficulty making code changes, especially since many changes involve just the content logic. Every time you change application code you typically need to recompile, link and redistribute the code as well as thoroughly test
30 the application. Additionally, when SQL statements are coded directly in code or as stored procedures the syntax used for parameters or variables typically need to

express the data types or those parameters. So even if the SQL itself doesn't change the code may have to if the data types for variables used in the code are changed. This indicates a need for a product that supports multiple SQL coding techniques and has runtime substitution of tokens including determining their data types but also
5 allows flexible design of SQL that can be easily put into stored procedures for performance reasons.

SUMMARY OF THE INVENTION

One embodiment of the present invention is the Tokenized SQL
10 Architecture ("TSA"), composed of three important technologies: the tokenized SQL file or .SQT file, the ExpandTokenizedSQL function and the CSP utility. Enhancing the development advantages of TSA is any template based rendering engine and authoring environment.

The tokenized SQL file or .SQT file is a file format that contains a
15 single SQL statement plus parameter and data type information (or tokens). This is a simple ASCII file using section and entry conventions similar to a MS Windows based .INI file. The SQT file may include an [SQL] section and a [TOKENS] section. In TSA terms the variables or parameters used in any SQL are Tokens since they represent both a parameter, it's run-time or design-time value as well as data type
20 information about the parameter. The embedded token syntax includes an @ symbol directly in front of the variable that you want expanded. This method was selected since it is identical to a stored procedure "named parameters" type syntax. Normally, the SQL is ready to run inside the target DBMS stored procedure environment as well as ready to run as dynamic SQL with the exception of specifying the parameter
25 values.

The SQL statement can easily be cut and paste into query editors or other development tools with valid values to supply to the parameters included in the file. No application code programming syntax such as concatenation operators or quotes need to be modified in order to run the statement. If used with the AccessVia

template authoring tool, no cut and paste operation is required. Important to making SQL easy to develop with is this tokenized SQL file or .SQT file.

The ExpandTokenizedSQL function can take a variety of SQL statement input sources, with the input type being declared by the source type parameter. Normally, in a development or design mode during the application development process, an SQT file name is provided with token names and values and the "File" source type. During production mode, this SQT file name will represent a stored procedure name and the appropriate parameter values specified along with the "Stored Procedure" source type. Other source types are also supported. See the function prototype in section 4.2 below.

The TSA includes an SQT2SP utility. This utility executable takes a folder of SQT files and generates all of the necessary stored procedure code for the database. This has been customized to use XML with DBMS specific XSL to easily support any new or existing DBMS. This utility reads an INI file that stores folder locations, a database connection string, and XSLT stylesheets to plug in at run time for the various database vendors. It will read in all of the SQT files in the source directory and process them. Optionally, it will read generate a new set of correct default values for entries in the [TOKENS] section. This insures that the default values stay in sync and up to date in the SQT file so that when a template author links to that SQT file it will run with current default values.

The TSA allows the code to be data type independent by pushing the SQL and data type specifications into the SQT file, without requiring difficult SQL parsing. The TSA specifies a matching table and column name reference for each parameter so that it's type can be easily tested. These entries are included in the [TOKENS] section. However, not all stored procedure parameters map directly to a table and column name so it is also an option to specify the specific data type. The table is opened to get the column info, which creates a performance hit, but in design mode this is fine, in production mode this will not be necessary as all of the data type information will have already been correctly generated for the stored procedure. The two purposes of these entries are to determine if quote characters need to be placed around the values for the appropriate parameter types and to generate the specific

data types necessary to generate the stored procedure call. Additionally, the [TOKENS] section entries can supply an actual or default value to be used with development tools like query editors.

The TSA supports more than one method of parameter calling to allow flexibility with an order or precedence. The ExpandTokenizedSQL function scans the SQL parameters and changes any quote provided to the correct quote required by the connected DBMS. For example, if the DBMS requires double quotes around string values and the user supplied single quotes then the single quotes are changed to double quotes. This makes the supplied SQL statement more portable though some parameters for SQL server that identify tables, columns or keywords will want to stay in double quotes. Quotes embedded in a value that are the same as the DBMS delimiting quote will now be escaped by doubling the value. If table and column names are provided in the [TOKENS] section then use them as described. If no tokens or quotes are provided then pass the parameter as is.

15

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a development process diagram.

Figure 2 illustrates moving a tokenized SQL to production mode.

Figure 3 illustrates tokenized SQL production deployment.

20

DETAILED DESCRIPTION OF THE INVENTION

1.1 SQT2SP utility usage

To convert SQT files to Stored Procedures run the SQT2SP utility by directing it to a folder of SQT files and a DSN. It will generate an output file that is ready to load all the appropriate stored procedures in the database server DSN specified. The developer then changes the global eSource variable in the application from the rddFile to the rddStoredProc value and the application will now run against the stored procedures. To support stored procedures, the SQT files need to have correctly specified entries. The [TOKENS] section should have all token names (without the leading @ sign) with each followed by an equal sign (=) and then a actual or sample

30

value that is formatted for the correct data type (' or " for strings, nothing for numbers). The token entry will be preceded by a table.columnname value that matches the token names correct data types, if no table.columnname combination is available for the variable then the DBMS specific data type is declared (like
 5 VarChar(20) or int). If the CreateSP.exe has trouble converting certain SQT files it is because they do not meet the [TOKENS] section entry requirements.

For each file, it will first convert the SQT file to an XML doc that is a "flat" XML document, meaning it has sections and rows elements only, no real structured content. This file is then converted to a "SQTDef" XML doc in memory using an xslt
 10 style sheet 'SQTFile2SQTDef.xslt'. This work is done in 2 parts: phase 1 does everything but the database lookup to get data types for the parameters. Phase 2 hits the database for the data types and converts from vendor-specific DBMS data types to OLEDB data types. This is a more robust XML schema. This will then be validated with another xslt style sheet, valSQTDef.xslt. It's a touch harder to validate
 15 with xslt than with code (xslt doesn't do substrings very well), it's also more maintainable without recompiling. The "transformed" SQTDef xml doc is a plain text memory stream that is then written out to a log file if there are errors. This uses xslt extension objects that call back from xslt to C#. If the SQTDef is valid, there is a final conversion to a stored procedure with a vendor-specific xslt style sheet, of
 20 which there is an SQL Server example provided. A different XSLT style sheet is for each different database vendor. All stored procedures can get written to a single file or optionally written to one stored procedure file per SQT file. These outputted stored procedures file can the loaded into a query editor or command environment for the database and executed, generating all of the stored procedures required for that
 25 application.

1.2 Expand Tokenized SQL prototype

THOMAS UPDATE HERE:

```

30  RDDAPI (RC)          ExpandTokenizedSQL(
      IN   DBC *        pDBC,
      IN   long         fSource,
      IN   pszSourceSQL,
      OUT  char *        pszExpandedSQL,
      OUT  long *        pcbExpandedSQL,
```

```

    IN    char *    pszTokenValues
    );

```

5	pDBC	Pointer to a DBC. This may be NULL for FILE and PARM sources.						
10	fSource be:	Defines the source of the SQL statement. May						
		<table border="0"> <tr> <td style="padding-right: 20px;">SQL_SOURCE_FILE</td> <td>Get the SQL from a file</td> </tr> <tr> <td>SQL_SOURCE_TABLE</td> <td>Get the SQL from a table</td> </tr> <tr> <td>SQL_SOURCE_PARM</td> <td>The SQL is passed in</td> </tr> </table>	SQL_SOURCE_FILE	Get the SQL from a file	SQL_SOURCE_TABLE	Get the SQL from a table	SQL_SOURCE_PARM	The SQL is passed in
SQL_SOURCE_FILE	Get the SQL from a file							
SQL_SOURCE_TABLE	Get the SQL from a table							
SQL_SOURCE_PARM	The SQL is passed in							
15	pszSQLSource procedure.	<table border="0"> <tr> <td style="padding-right: 20px;">SQL_SOURCE_PROC</td> <td>The SQL is a stored</td> </tr> </table>	SQL_SOURCE_PROC	The SQL is a stored				
SQL_SOURCE_PROC	The SQL is a stored							
	pszSourceSQL SQL	String containing filename, record key or						
20	pszExpandedSQL statement. may	Pointer to a buffer for expanded SQL This buffer is allocated by the caller. You						
25	NULL and in	call the function with this parameter set to the required buffer length will be returned in pcbExpandedSQL.						
30	pcbExpandedSQL	Length of the expanded SQL statement buffer.						
	pszTokenValues	Pointer to Key=value; pairs for tokens. Each key=value pair needs to be terminated						
35	by a semi-colon.							

Expand Tokenized SQL function usage

ExpandTokenizedSQL function calls returns an executable SQL statement string with any string tokens replaced by values specified in TokenValues parameter. Or it the stored procedure name then parameter list and this stored procedure parameter list will include the 'single quote' for the string parameter, date parameter etc since the stored procedures expects single quote with strings and date type input parameter list. For eSource = rddTable, the ActiveConnection property needs to be set with an active connection object. For eSource = rddFile, the SourceSQL can either be a fully qualified file name or file name with no path information. For a file name with no path information, the object assumes the file is located in the current vDir.

```

rc = rddExpandTokenizedSQL(pDBC,
    SQL_SOURCE_FILE | SQL_SOURCE_PROC,

```

```

        "\ADirectory\AFile.sqt",
        &pszSQLBuffer,
        &cbSQLBuffer,
        "strOrgName=SIGN & strSignType=TYPE"
5         );

```

TokenValues parameter specifies the list of token/value pairs to replace tokenized parameters in the SourceSQL. This parameter string needs to be in the format:

```

token=value                                for a single token/value pair
or
10 token(1)=value(1)&token(2)=value(2)&...&token(n)=value(n)    for n token value pairs

```

The eSource parameter is an enumerated value which specifies the SQLSource type. This value is defaulted to rddString and is not required. Get the SQL source from the passed in parameter, a file, SGS SQL table record for just build the stored procedure call. Possible eSource values are:

```

rddFile           the SQLSource is the name of a file or a fully qualified file
name.
rddStoredProc     the SQLSource is the name of a stored procedure
rddString         the SQLSource is a SQL statement string
20 rddTable        the SQLSource is the name of a primary key value in SGS SQL

```

Null string values are accepted by rddExpandTokenized SQL. When passing in the substitution values in the pszTokenValues parameter DO NOT pass in token value pairs that have no value and allow the default token to assign the value. Then when rddExpandTokenizedSQL is called pass in the tokens that have actual values, the others will be set to NULL by default if the NULL is specified in the [TOKENS] section for the token entry. This way SQL UPDATE or INSERT statements with tokens that may or may not have values can be used. Putting in two double quotes and double quotes around a space is difficult to read and may lose that token and all subsequent tokens. This way the ExpandTokenizedSQL function will provide the necessary values and something in the statement to indicate "this field empty" to the DB.

For example:

```

[TOKENS]
BUS_SCHEDULE.BUS_ROUTE BusRoute = NULL

```

35

Function Internals:

1. Check all parameters for valid values.
2. Read the tokenized SQL source from a file, table, or string buffer
3. If the source is in a file or table the [TOKENS] entry may be included in the source
If they are present then read and concatenate with pszTokenValues.
4. Expanded the tokenized SQL, substituting token values where supplied

40

1.3 Sample database table named BUS_SCHEDULE

BUS_ROUTE	ROUTE_DIRECTION	BUS_STOP	ARRIVAL_TIME
NUMERIC(4)	VARCHAR(20)	VARCHAR(40)	DATETIME
8	Southbound	15 th Ave and 80 St	8:07 AM
8	Southbound	15 th Ave and 65 St	8:10 AM
8	Southbound	15 th Ave and 45 St	8:13 AM
8	Southbound	15 th Ave and Main St	8:20 AM

1.4 Example SQL program without TSA

'Initialize the MS ADO connection object and open a new connection

Dim oConn as New ADODB.Connection

```
5 oConn.Open "driver={SQL Server};server=DB_SERVER;user
id=sa;password=;database=BUS_SCHEDULE;"
```

'Initialize the recordset object and open a new recordset using the connection and SQL statement

```
10 Dim rsBus as New ADODB.RecordSet
```

'Build the SQL Statement using variables for the bus route

'normally these variables are determined by some user interface calls

Dim numBusRoute = 8

```
15 Dim strRouteDirection = "Southbound"
```

'note the included quotes around the RoutDirection variable

Dim strSQLstatement = "SELECT * FROM BUS_SCHEDULE WHERE BUS_ROUTE = " & numBusRoute & " ROUTE_DIRECTION = " & strRouteDirection & ""

```
20 Set rsBus = oConn.Execute(strSQLstatement)
```

'THE REST IS THE SAME FOR ALL EXAMPLES

'walk the recordset, printing the route, stop and arrival time

While Not rsBus.EOF

```
25 Debug.Print "Route: #" & rsBus("BUS_ROUTE") & " " & rsBus("ROUTE_DIRECTION") & "
Stop: " & rsBus("BUS_STOP") & " Time: " & rsBus("ARRIVAL_TIME")
```

rsBus.MoveNext

Wend

rsBus.Close

```
30 oConn.Close
```

1.5 Sample Tokenized SQL (SQT) file used with example

BUS_ROUTE.SQT

[SQL]

5 SELECT * FROM BUS_SCHEDULE
WHERE BUS_ROUTE = @BusRoute AND ROUTE_DIRECTION = @RouteDirection

[TOKENS]

BUS_SCHEDULE.BUS_ROUTE BusRoute= 0
BUS_SCHEDULE.ROUTE_DIRECTION RouteDirection = 'Northbound'

[COMMENTS]

10 select statement for use with printing a bus schedule

1.6 Example SQL program with SQT file

'Initialize the AccessVia database connection object and open a new connection

Dim oConn as New RDD.Connection

15 oConn.Open "driver={SQL Server};server=DB_SERVER;user
id=sa;password=;database=BUS_SCHEDULE;"

'Initialize the command object to parse the SQT file and open a new recordset using the connection and SQL statement

Dim oCommand as New RDD.Command

20

'Build the SQL Statement from the SQT file using variables for the bus route

'normally these variables are determined by some user interface calls

Dim numBusRoute = 8

Dim strRouteDirection = "Southbound"

25

'normally this eSource variable is defined globally from a configuration table or file

Dim eSource = rddFile 'source for SQT statement is from file, not string or stored procedure

'note that no quotes are required around the route direction value

30 Dim vTokens = "BusRoute=" & numBusRoute & "&RouteDirection=" & strRouteDirection

Dim strSQLstatement =

oCommand.ExpandTokenizedSQL("BUS_ROUTE.SQT",vTokens,eSource)

oCommand.Close

35 'Initialize the recordset object and open a new recordset using the connection and SQL statement

Dim rsBus as New RDD.RecordSet

Set rsBus = oConn.Execute(strSQLstatement)

40 'THE REST IS THE SAME AS EXAMPLE 5.5

'walk the recordset, printing the route, stop and arrival time

While Not rsBus.EOF

 Debug.Print "Route: #" & rsBus("BUS_ROUTE") & " " & rsBus("ROUTE_DIRECTION") & "
Stop: " & rsBus("BUS_STOP") & " Time: " & rsBus("ARRIVAL_TIME")

 rsBus.MoveNext

5 Wend

 rsBus.Close

 oConn.Close

1.7 Example Stored Procedure program without TSA

10 'Initialize the MS ADO connection object and open a new connection

 Dim oConn as New ADODB.Connection

 oConn.Open "driver={SQL Server};server=DB_SERVER;user
id=sa;password=;database=BUS_SCHEDULE;"

15 ' Open a command object for a stored procedure with two parameters, note this code is
unique

 Dim cmdBus as New ADODB.Command

 Set cmdBus.ActiveConnection = oConn

 cmdBus.CommandText = "BUS_ROUTE"

 cmdBus.CommandType = adCmdStoredProc

20 cmdBus.Parameters.Refresh

 ' Get parameter value, execute the command and store the results in a recordset

 'normally these variables are determined by some user interface calls

 cmdBus.Parameters(1) = 8

25 cmdBus.Parameters(2) = "Southbound"

 'Initialize the recordset object and open a new recordset using the command object

 Set rsBus = cmdBus.Execute()

30 'THE REST IS THE SAME AS EXAMPLE 5.5

 'walk the recordset, printing the route, stop and arrival time

 While Not rsBus.EOF

 Debug.Print "Route: #" & rsBus("BUS_ROUTE") & " " & rsBus("ROUTE_DIRECTION") & "
Stop: " & rsBus("BUS_STOP") & " Time: " & rsBus("ARRIVAL_TIME")

35 rsBus.MoveNext

 Wend

 rsBus.Close

 oConn.Close

1.8 Sample Stored Procedure used with example

40 CREATE PROCEDURE BUS_ROUTE (@BusRoute as Numeric(2), @RouteDirection as
VarChar(20))

AS

BEGIN

SELECT * FROM BUS_SCHEDULE
WHERE BUS_ROUTE = @BusRoute AND ROUTE_DIRECTION = @RouteDirection

5 END

GO

1.9 Example Stored Procedure program with TSA

'Initialize the AccessVia database connection object and open a new connection

Dim oConn as New RDD.Connection

10 oConn.Open "driver={SQL Server};server=DB_SERVER;user
id=sa;password=;database=BUS_SCHEDULE;"

'Initialize the command object to parse the SQT file and open a new recordset using the
connection and SQL statement

15 Dim oCommand as New RDD.Command

'Build the SQL Statement from the SQT file using variables for the bus route

'normally these variables are determined by some user interface calls

Dim numBusRoute = 8

20 Dim strRouteDirection = "Southbound"

'normally this eSource variable is defined globally from a configuration table or file

Dim eSource = rddStoredProc 'source for SQT statement is from stored procedure

'THE ABOVE LINE IS THE ONLY DIFFERENCE FROM EXAMPLE 5.7

25

Dim vTokens = "BusRoute=" & numBusRoute & "&RouteDirection=" & strRouteDirection

Dim strSQLstatement =

oCommand.ExpandTokenizedSQL("BUS_ROUTE.SQT",vTokens,eSource)

oCommand.Close

30

'Initialize the recordset object and open a new recordset using the connection and SQL
statement

Dim oRS as New RDD.RecordSet

Set rsBus = oConn.Execute(strSQLstatement)

35

'THE REST IS THE SAME AS EXAMPLE 5.5

'walk the recordset, printing the route, stop and arrival time

While Not rsBus.EOF

40 Debug.Print "Route: #" & rsBus("BUS_ROUTE") & " " & rsBus("ROUTE_DIRECTION") & "
Stop: " & rsBus("BUS_STOP") & " Time: " & rsBus("ARRIVAL_TIME")

```

        rsBus.MoveNext
    Wend
    rsBus.Close
    oConn.Close

```

5

1.10 Source Code and XSLT for SQT2SP utility

The following source code is used to take an SQT file and a database connection, and generate database specific stored procedures

1.10.1 MS C# Source Code for SQT2SP executable

10

```

using System;
using System.Drawing;
using System.Collections;
using System.Collections.Specialized;
15 using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Xml;
using System.Xml.Xsl;
20 using System.IO;
using System.Data.OleDb;
using System.Diagnostics;
using System.Text.RegularExpressions;
/* non-MS namespaces */
25 using IniFile;

namespace SQT2StoredProc
{
    /// <summary>
    30    /// Summary description.
    /// </summary>
    public class SQT2XMLTransform
    {
        /* fields */
        35        private string m_SQTSourceDir;
        private string m_StoredProcOutputDir;
        private string m_StoredProcOutputFile = "CreateSQTStoredProcs.sql";
        //private string m_StoredProcOutputFile = "";
        40        private string m_XsltDir;
        private string m_SQTFile2SQTDefXslt = "SQTFile2SQTDef.xslt";
        private string m_SQTDef2StoredProcXslt;
        protected XslTransform m_SQTFile2SQTDefTransform;
        protected XslTransform m_SQTDef2StoredProcTransform;
        45        protected XslTransform m_SQTDefValidator;
        private OleDbConnection m_dbConnection;
        protected string m_connectionString =
            "Provider=SQLOLEDB;Data Source=localhost;Initial
            Catalog=WDSS;Integrated Security=SSPI;";
        50        /* properties */

        public string SQTSourceDir
        {
            55            get {return m_SQTSourceDir;}
            set {m_SQTSourceDir = value;}
        }
        public string StoredProcOutputDir

```

```

    {
        get {return m_StoredProcOutputDir;}
        set {m_StoredProcOutputDir = value;}
    }
5   public string StoredProcOutputFile
    {
        get {return m_StoredProcOutputFile;}
        set {m_StoredProcOutputFile = value;}
    }
10  public string XsltDir
    {
        get {return m_XsltDir;}
        set {m_XsltDir = value;}
    }
15  public string SQTFile2SQTDefXslt
    {
        get {return m_SQTFile2SQTDefXslt;}
        set {m_SQTFile2SQTDefXslt = value;}
    }
20  public string SQTDef2StoredProcXslt
    {
        get {return m_SQTDef2StoredProcXslt;}
        set {m_SQTDef2StoredProcXslt = value;}
    }
25  public string ConnectionString
    {
        get {return m_connectionString;}
        set {m_connectionString = value;}
    }
30

    /* Constructors */

    public SQT2XMLTransform(string inSQTSourceDir, string
35  inStoredProcOutputDir, string inXsltDir, string inSQTDef2StoredProcXslt,
    string inSQTDefValidator, string inConnectionString)
    {
        try
        {
40            m_SQTSourceDir = inSQTSourceDir;
            StoredProcOutputDir = inStoredProcOutputDir;
            XsltDir = inXsltDir;
            ConnectionString = inConnectionString;
            m_SQTDef2StoredProcXslt = inSQTDef2StoredProcXslt;
45            m_SQTFile2SQTDefTransform = new XslTransform();
            m_SQTFile2SQTDefTransform.Load(XsltDir + SQTFile2SQTDefXslt);
            m_SQTDefValidator = new XslTransform();
            m_SQTDefValidator.Load(XsltDir + inSQTDefValidator);
            m_SQTDef2StoredProcTransform = new XslTransform();
50            m_SQTDef2StoredProcTransform.Load(XsltDir + SQTDef2StoredProcXslt);
            m_dbConnection = new OleDbConnection();
        }
        catch(Exception e)
        {
55            Trace.WriteLine(System.DateTime.Now.ToString("F") + " :
SQT2StoredProc: " + e.Message + e.StackTrace);
            Application.Exit();
        }
    }
60

    public static void Main(string[] args)
    {

        /* Set up error logging */
65        Trace.Listeners.Add(new
            TextWriterTraceListener(File.Create(Application.StartupPath + @"\" +
"SQT2StoredProc.log")));
        Trace.AutoFlush = true;
    }

```

```

        Trace.Write("");
        Trace.Indent();

        Trace.WriteLine(System.DateTime.Now.ToString("F") + ": SQT2StoredProc:
5 Program Start");
        /* Read INI file from command line. Default is SQT2StoredProc.INI in
        same folder as exe file */
        string strINIFileDir;
        if (args.Length == 0)
10         strINIFileDir = Application.StartupPath + @"\" +
        "SQT2StoredProc.INI";
        else
            strINIFileDir = args[0];
        IniFileReader ifr = new IniFileReader(strINIFileDir, true);
15         /* Read INI file entries and instantiate a transformer class */
        string strSQTSourceDir = ifr.GetIniValue("Startup",
        "SQTSourceFolder");
        if (!strSQTSourceDir.EndsWith(@"\"))
            strSQTSourceDir = strSQTSourceDir + @"\";
20         string strStoredProcOutputDir = ifr.GetIniValue("Startup",
        "StoredProcOutputFolder");
        string strXsltFolder = ifr.GetIniValue("Startup", "XsltFolder");
        string strSQTDef2StoredProcXsltFileName = ifr.GetIniValue("Startup",
        "SQTDef2StoredProcXsltFileName");
25         string strDBConnectionString = ifr.GetIniValue("Startup",
        "DBConnectionString");
        string strSQTDefValidator = ifr.GetIniValue("Startup",
        "SQTDefValidatorXsltFileName");
        SQT2XMLTransform transformer =
30         new SQT2XMLTransform(strSQTSourceDir,
            strStoredProcOutputDir,
            strXsltFolder,
            strSQTDef2StoredProcXsltFileName,
            strSQTDefValidator,
35             strDBConnectionString);
        transformer.Transform();
        Trace.Unindent();
        Trace.WriteLine(System.DateTime.Now.ToString("F") + ": SQT2StoredProc:
        Program End");
40     }
    public bool Transform()
    {
        /*
        * Process the source directory in a loop, reading each SQT file and
45     transforming
        * first, instantiate objects and do some setup work outside the loop,
        though.
        */
        try
50     {
            DirectoryInfo SQTDirInfo = new DirectoryInfo(SQTSourceDir);
            //FileInfo[] SQTFileList = SQTDirInfo.GetFiles("*.SQT");
            //FileInfo[] SQTFileList = SQTDirInfo.GetFiles("Delete*.SQT");
            //FileInfo[] SQTFileList =
55     SQTDirInfo.GetFiles("DeleteSelectedSignStoreSignIDs.SQT");
            FileInfo[] SQTFileList = SQTDirInfo.GetFiles("AddMemberL1.SQT");
            //FileInfo[] SQTFileList =
            SQTDirInfo.GetFiles("AddCopySignsForBatch.SQT");
            XmlDocument SQTDefXmlDoc = new XmlDocument();
60         MemoryStream SQTDefMemStream = new MemoryStream();
            FileStream StoredProcSingleStream =
                new FileStream(StoredProcOutputDir + StoredProcOutputFile,
                FileMode.Create,
                FileAccess.Write,
65         FileShare.Read);
            StreamWriter StoredProcOutWriter = new
            StreamWriter(StoredProcSingleStream);
            OpenDbConnection(this.ConnectionString);

```

```

        foreach (FileInfo SQTFile in SQTFileList)
        {
            /*
5          * First reads the INI-file-like SQT file from disk and convert it
to a          * hierarchical, schema-based XML format in a memory stream.
            */
            Trace.WriteLine(System.DateTime.Now.ToString("F") + ":
10      SQT2StoredProc: Processing file " + SQTFile.Name);
            Trace.Indent();
            ConvertSQTFileToXML(SQTSourceDir + SQTFile.Name, ref
SQTDefMemStream);
            /*
15          * Load the stream to an XmlDocument, then update the parameter
datatypes in  * the SQTDef with information from the target database using
OleDb.          */
20          SQTDefXmlDoc.Load(SQTDefMemStream);
            UpdateParameterDataTypes(ref SQTDefXmlDoc);

            //SQTDefXmlDoc.Save("E:\\data\\Source\\vbs\\SQT2SP\\sqtFiles\\SQTDef.XML");
            /*
25          * Edit SQTDef xml document using XSLT instead of schemas, so we
can control    * the processing and error messages, and write to a log file. If
there are errors,
            * skip to the next file.
            */
30          if (!SQTDefIsValid(ref SQTDefXmlDoc))
            {
                Trace.Unindent();
                continue;
35          }
            /*
            * The SQTDef is now complete. Transform it to actual stored
procedure code
            * for the target database. The XSLT stylesheet outputs to a non-
40      XML stream.
            */
            MemoryStream StoredProcMemStream = new MemoryStream();
            m_SQTDef2StoredProcTransform.Transform(SQTDefXmlDoc, null,
StoredProcMemStream);
45          StoredProcMemStream.Position = 0;
            /*
            */
            /*
50          if (StoredProcOutputFile == "")
            {
                string[] outFileName = SQTFile.Name.Split('.');
                outFileName[1] = ".SQL";
                FileStream StoredProcOutStream =
                new FileStream(StoredProcOutputDir + outFileName[0] + "." +
55      outFileName[1],
                FileMode.Create,
                FileAccess.Write,
                FileShare.Read);
                /*
60          * Output the stream to persistent storage on disk */
                StreamReader StoredProcReader = new
StreamReader(StoredProcMemStream);
                while (StoredProcReader.Peek() > -1)
                {
50                  StoredProcOutWriter.WriteLine(StoredProcReader.ReadLine());
                }
                StoredProcReader.Close();
                if (StoredProcOutputFile == "")

```



```

//      {
//      StoredProcOutputStream.Close();
//      }
5      Trace.Unindent();
    }
    StoredProcOutWriter.Close();
    CloseDbConnection();
    } // end try block
    catch(Exception e)
10    {
        Trace.WriteLine(System.DateTime.Now.ToString("F") + ":
SQT2StoredProc: " + e.Message + e.StackTrace);
    }

15    return true;
    }
    protected void OpenDbConnection(string inConnectionString)
    {
20        m_dbConnection.ConnectionString = ConnectionString;
        try
        {
            m_dbConnection.Open();
        }
        catch(Exception e)
25        {
            Trace.WriteLine(System.DateTime.Now.ToString("F") + ":
SQT2StoredProc: " + e.Message + e.StackTrace);
        }
    }
30    protected void CloseDbConnection()
    {
        m_dbConnection.Close();
    }
    protected void ConvertSQTFileToXML(string inFile, ref MemoryStream
35    inStream)
    {
        try
        {
40            /*
             * Create an XML document from the SQT file. The schema for this
            document is a general
             * INI-file-like schema with sections, and items that have 'key' and
            'value' attributes.
             */
45            IniFileReader ifr = new IniFileReader(inFile, true);
            /* Uncomment the following statements as needed for debugging.
             *
             * To Save to a File:
            */
50            //ifr.OutputFilename = "<your output.XML>";
            //ifr.Save();
            /*
             * To Display:
            */
55            //MessageBox.Show(ifr.XML);
            /*
             * SQTFile2SQTDef stylesheet takes the base of the filename as a
            parameter.
             * This will eventually become the name of the stored procedure.
60            */
            XsltArgumentList xsltArgs = new XsltArgumentList();
            FileInfo fileInfo = new FileInfo(inFile);
            string fileName = fileInfo.Name;
            string[] baseName = fileName.Split('.');
65            xsltArgs.AddParam("inSQTName", "", baseName[0]);
            /* Make sure to reset the stream properly for reuse in the loop:
            shrink its

```

```

        * length to zero and put the stream pointer at the beginning BEFORE
processing,
        * and set the position back to zero AFTER transforming. This is
important!
5      */
        inStream.SetLength(0);
        inStream.Seek(0, SeekOrigin.Begin);
        m_SQTFFile2SQTFDefTransform.Transform(ifr.XmlDoc, xslArgs, inStream);
10     }
        inStream.Position = 0; // do this so the stream can be read later
    }
    catch(Exception e)
    {
        Trace.WriteLine(System.DateTime.Now.ToString("F") + ":
SQT2StoredProc: e.Message + e.StackTrace");
15    }
}

protected void UpdateParameterDataTypes(ref XmlDocument inXmlDoc)
{
20    OleDbType OleDbTypeInstance = new OleDbType();
    string strOleDbDataType = "";
    short shortNumericScale = 0;
    int intNumericPrecision = 0;
    long lDatetimePrecision = 0;
25    long lCharMaxLength = 0;
    long lColumnFlags = 0;
    /* const values DBCOLUMNFLAGSENUM enum in OleDb.h; couldn't figger out
how to include in
    * C#, what class they belong to, etc. I hope they don't change!
30    */
    const long lIsLong = 0x80;
    const long lIsFixedLength = 0x10;
    const long lScaleIsNegative = 0x4000;
    long fixedLengthTest;
35    long longTest;
    long ScaleIsNegativeTest;

    XmlNodeList SQLColumnRefNodes =

40    inXmlDoc.DocumentElement.SelectNodes("SQTParameterList/SQTPParameter/SQLColumn
nRef");
    foreach(XmlNode SQLColumnRefNode in SQLColumnRefNodes)
    {
        string strTableName =
45    SQLColumnRefNode.SelectSingleNode("SQLTable").Attributes["name"].Value;
        string strColumnName =
        SQLColumnRefNode.SelectSingleNode("SQLColumn").Attributes["name"].Value;
        Object[] columnFilter = {null, null, strTableName, strColumnName};
        /* Now get schema information from the database itself. */
50    DataTable myColumnInfo =
        m_dbConnection.GetOleDbSchemaTable(OleDbSchemaGuid.Columns, columnFilter);
        foreach(DataRow myRow in myColumnInfo.Rows)
        {
            strOleDbDataType =
55    Enum.GetName(OleDbTypeInstance.GetType(), myRow["DATA_TYPE"]);
            if (myRow["NUMERIC_SCALE"] != System.DBNull.Value)
                shortNumericScale = (short) myRow["NUMERIC_SCALE"];
            if (myRow["NUMERIC_PRECISION"] != System.DBNull.Value)
                intNumericPrecision = (int) myRow["NUMERIC_PRECISION"];
60            if (myRow["DATETIME_PRECISION"] != System.DBNull.Value)
                lDatetimePrecision = (long) myRow["DATETIME_PRECISION"];
            if (myRow["CHARACTER_MAXIMUM_LENGTH"] != System.DBNull.Value)
                lCharMaxLength = (long) myRow["CHARACTER_MAXIMUM_LENGTH"];
            if (myRow["COLUMN_FLAGS"] != System.DBNull.Value)
65            lColumnFlags = (long) myRow["COLUMN_FLAGS"];
        }
        fixedLengthTest = lColumnFlags & lIsFixedLength;
        longTest = lColumnFlags & lIsLong;

```

```

        ScaleIsNegativeTest = lColumnFlags & lScaleIsNegative;
        /* Update OleDbDataType attributes based on what was found in the
database */
        XmlNode dataTypeNode =
5      SQLColumnRefNode.SelectSingleNode("SQLColumn/OleDbDataType");
        dataTypeNode.Attributes["name"].Value = strOleDbDataType;
        if (shortNumericScale > 0)
            dataTypeNode.Attributes["numericScale"].Value =
shortNumericScale.ToString();
10         if (intNumericPrecision > 0)
            dataTypeNode.Attributes["numericPrecision"].Value =
intNumericPrecision.ToString();
            if (lDatetimePrecision > 0)
                dataTypeNode.Attributes["dateTimePrecision"].Value =
15      lDatetimePrecision.ToString();
            if (lCharMaxLength > 0)
                dataTypeNode.Attributes["characterLength"].Value =
lCharMaxLength.ToString();
            if (fixedLengthTest == 0x10)
20         dataTypeNode.Attributes["isFixedLength"].Value = "true";
            else
                dataTypeNode.Attributes["isFixedLength"].Value = "false";
            if (longTest == 0x80)
                dataTypeNode.Attributes["isLong"].Value = "true";
25         else
            dataTypeNode.Attributes["isLong"].Value = "false";
            if (ScaleIsNegativeTest == 0x4000)
                dataTypeNode.Attributes["scaleIsNegative"].Value = "true";
            else
30         dataTypeNode.Attributes["scaleIsNegative"].Value = "false";
        /* clear values */
        lCharMaxLength = 0;
        shortNumericScale = 0;
        intNumericPrecision = 0;
35      lDatetimePrecision = 0;
        fixedLengthTest = 0;
        longTest = 0;
        ScaleIsNegativeTest = 0;
    }
40    //MessageBox.Show(inXmlDoc.OuterXml);
}

protected bool SQTDefIsValid(ref XmlDocument inXmlDoc)
{
45    MemoryStream errorLogStream = new MemoryStream();
    XsltArgumentList xslArgs = new XsltArgumentList();
    xslArgs.AddExtensionObject("urn:SQT2StoredProc", this);
    m_SQTDefValidator.Transform(inXmlDoc, xslArgs, errorLogStream);
    if (errorLogStream.Length > 3)
50    {
        /* Output the stream to the log file */
        StreamReader sr = new StreamReader(errorLogStream);
        errorLogStream.Position = 0;
        while (sr.Peek() > -1)
55        {
            Trace.WriteLine(sr.ReadLine());
        }
        sr.Close();
        return false;
60    }
    else
        return true;
}

65 public string paramNameIsValid(string inString)
{
    Regex regex = new Regex("[a-zA-Z][a-zA-Z0-9]*");

```

```

        return regex.Match(inString).Value;
    }
}
}

```

5 1.10.2 SQT2SP XSLT XML documents

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 2 U (http://www.xmlspy.com) by William
Barnum (Personal Copy) -->
<!-- =====>
10 <!-- Internal general entities for boilerplate -->
<!-- =====>
<!DOCTYPE xsl:stylesheet [
<!ENTITY newLine "&#xd;&#xa;">
]>
15 <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" version="1.0" encoding="UTF-8" indent="no"/>
  <!-- =====>
  <!-- key lookups -->
20 <!-- =====>
  <!-- no keys in this styleteet -->
  <!-- =====>
  <!--global variables -->
  <!-- =====>
25 <!-- no globals in this styleteet -->
  <!-- =====>
  <!-- root template -->
  <!-- =====>
  <xsl:template match="/">
30   <xsl:text>CREATE PROCEDURE </xsl:text>
   <xsl:value-of select="/SQTDef/@name"/>
   <xsl:text> (</xsl:text>
   <xsl:call-template name="CreateParameterList"/>
   <xsl:text>)&newline;</xsl:text>
35   <xsl:text>AS&newline;</xsl:text>
   <xsl:text>BEGIN&newline;</xsl:text>
   <xsl:call-template name="CreateSQLStmtList"/>
   <xsl:text>END&newline;</xsl:text>
   <xsl:text>GO&newline;</xsl:text>
40 </xsl:template>
  <!-- =====>
  <!-- CreateParameterList-->
  <!-- =====>
  <xsl:template name="CreateParameterList">
45   <xsl:for-each select="//SQTParameter">
    <xsl:text>&#x40;</xsl:text>
    <xsl:value-of select="@name"/>
    <xsl:text> AS </xsl:text>
    <xsl:call-template name="OleDbDataType2SQLServer">
50      <xsl:with-param name="inSQLColumn" select="SQLColumnRef/SQLColumn"/>
    </xsl:call-template>
    <xsl:if test="position() != last()">, </xsl:if>
  </xsl:for-each>
  </xsl:template>
55 <!-- =====>
  <!-- OleDbDataType2SQLServer -->
  <!-- =====>
  <xsl:template name="OleDbDataType2SQLServer">
    <xsl:param name="inSQLColumn"/>
60    <xsl:variable name="vName" select="$inSQLColumn/OleDbDataType/@name"/>
    <xsl:variable name="vNumericScale"
select="$inSQLColumn/OleDbDataType/@numericScale"/>
    <xsl:variable name="vNumericPrecision"
select="$inSQLColumn/OleDbDataType/@numericPrecision"/>
65    <xsl:variable name="vDateTimePrecision"
select="$inSQLColumn/OleDbDataType/@dateTimePrecision"/>

```

```

    <xsl:variable name="vCharacterLength"
select="$inSQLColumn/OleDbDataType/@characterLength"/>
    <xsl:variable name="vIsFixedLength"
select="$inSQLColumn/OleDbDataType/@isFixedLength"/>
5    <xsl:variable name="vIsLong"
select="$inSQLColumn/OleDbDataType/@isLong"/>
    <xsl:variable name="vScaleIsNegative"
select="$inSQLColumn/OleDbDataType/@scaleIsNegative"/>
    <xsl:choose>
10    <xsl:when test="$vName='Char'">
        <xsl:choose>
            <xsl:when test="$vIsFixedLength='true'">
                <xsl:value-of select="concat('char(', $vCharacterLength, '))'"/>
            </xsl:when>
15    <xsl:when test="$vIsLong='false'">
                <xsl:value-of select="concat('varchar(', $vCharacterLength,
                '))'"/>
            </xsl:when>
            <xsl:otherwise>text</xsl:otherwise>
20    </xsl:choose>
        </xsl:when>
        <xsl:when test="$vName='WChar'">
            <xsl:choose>
25    <xsl:when test="$vIsFixedLength='true'">
                <xsl:value-of select="concat('nchar(', $vCharacterLength,
                '))'"/>
            </xsl:when>
            <xsl:when test="$vIsLong='false'">
                <xsl:value-of select="concat('nvarchar(', $vCharacterLength,
30    <'')'"/>
            </xsl:when>
            <xsl:otherwise>ntext</xsl:otherwise>
            </xsl:choose>
        </xsl:when>
35    <xsl:when test="$vName='Integer'">int</xsl:when>
        <xsl:when test="$vName='BigInt'">bigint</xsl:when>
        <xsl:when test="$vName='DBTimeStamp'">datetime</xsl:when>
        <xsl:when test="$vName='Boolean'">bit</xsl:when>
        <xsl:when test="$vName='Numeric'">
40    <xsl:variable name="vTempScale">
            <xsl:choose>
                <xsl:when test="$vNumericScale='none'">0</xsl:when>
                <xsl:otherwise><xsl:value-of
select="$vNumericScale"/></xsl:otherwise>
45    </xsl:choose>
            </xsl:variable>
            <xsl:value-of select="concat('numeric(', $vNumericPrecision, ',',
$vTempScale, '))'"/>
        </xsl:when>
50    <xsl:when test="$vName='Binary'">
            <xsl:choose>
                <xsl:when test="$vIsFixedLength='true'">
                    <xsl:choose>
55    <xsl:when test="$vCharacterLength='8'">timestamp</xsl:when>
                    <xsl:otherwise>
                        <xsl:value-of select="concat('binary(', $vCharacterLength,
                        '))'"/>
                    </xsl:otherwise>
                    </xsl:choose>
                </xsl:choose>
60    </xsl:when>
            <xsl:otherwise>
                <xsl:choose>
                    <xsl:when test="$vIsLong='true'">image</xsl:when>
                    <xsl:otherwise>
65    <xsl:value-of select="concat('varbinary(',
$vCharacterLength, '))'"/>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:choose>

```

```

        </xsl:otherwise>
    </xsl:choose>
</xsl:when>
<xsl:when test="$vName='Currency'">
5    <xsl:choose>
        <xsl:when test="$vNumericPrecision='19'">money</xsl:when>
        <xsl:otherwise>smallmoney</xsl:otherwise>
    </xsl:choose>
</xsl:when>
10 <xsl:when test="$vName='Double'">float</xsl:when>
    <xsl:when test="$vName='Single'">
        <xsl:choose>
            <xsl:when
15 test="$vNumericPrecision='none'">smalldatetime</xsl:when>
            <xsl:otherwise>real</xsl:otherwise>
        </xsl:choose>
    </xsl:when>
    <xsl:when test="$vName='SmallInt'">smallint</xsl:when>
    <xsl:when test="$vName='UnsignedTinyInt'">tinyint</xsl:when>
20 <xsl:when test="$vName='Variant'">sql_variant</xsl:when>
    <xsl:when test="$vName='Guid'">uniqueid</xsl:when>
    <xsl:otherwise>
        <xsl:value-of select="concat('Unknown DataType: ', $vName)"/>
    </xsl:otherwise>
25 </xsl:choose>
</xsl:template>
<!-- =====>
<!-- CreateSQLStmtList -->
<!-- =====>
30 <xsl:template name="CreateSQLStmtList">
    <xsl:for-each select="//SQLStmt">
        <xsl:value-of select="@text"/>
        <xsl:text>&newline;</xsl:text>
    </xsl:for-each>
35 </xsl:template>
<!-- =====>
<!-- =====>
<!-- =====>
40 </xsl:stylesheet>

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 2 U (http://www.xmlspy.com) by William
Barnum (Personal Copy) -->
45 <!-- =====>
<!-- Internal general entities for boilerplate -->
<!-- =====>
<!DOCTYPE xsl:stylesheet [
<!ENTITY newLine "&#xd;&#xa;">
50 ]>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:param name="inSQTName" select="'Unknown'"/>
55 <!-- =====>
<!-- key lookups -->
<!-- =====>
    <xsl:key name="keyColumnRef"
match="sections/section[@name='COMMENTS']/item" use="@key"/>
60 <!-- =====>
<!--global variables -->
<!-- =====>
<!-- no globals in this styleteet -->
<!-- =====>
65 <!-- root template -->
<!-- =====>
    <xsl:template match="/">
        <SQTDef>

```

```

        <xsl:attribute name="name"><xsl:value-of
select="$inSQTName"/></xsl:attribute>
        <xsl:element name="SQTParameterList">
          <xsl:call-template name="SQTParameterList"></xsl:call-template>
5      </xsl:element>
        <xsl:element name="SQLStmtList">
          <xsl:call-template name="SQLStmtList"></xsl:call-template>
        </xsl:element>
      </SQTDef>
10    </xsl:template>
    <!-- =====>
    <!-- SQTParameterList                                -->
    <!-- =====>
    <xsl:template name="SQTParameterList">
15    <xsl:for-each select="sections/section[@name='TOKENS' or @name='DEFAULT
TOKENS']/item">
      <xsl:element name="SQTParameter">
        <xsl:call-template name="SQTParameter">
          <xsl:with-param name="inKey" select="@key"/>
20      </xsl:call-template>
    </xsl:element>
  </xsl:for-each>
</xsl:template>
    <!-- =====>
25    <!-- SQTParameter                                -->
    <!-- =====>
    <xsl:template name="SQTParameter">
      <xsl:param name="inKey"/>
      <xsl:variable name="varParamName">
30      <xsl:choose>
        <xsl:when test="starts-with(normalize-space($inKey), 'NUMBER') or
starts-with(normalize-space($inKey), 'STRING') ">
          <xsl:value-of select="substring-after(normalize-space($inKey), '
')"/>
35      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="normalize-space($inKey)"/>
      </xsl:otherwise>
    </xsl:choose>
40    </xsl:variable>
    <xsl:variable name="varParamDataType">
      <xsl:choose>
        <xsl:when test="starts-with(normalize-space($inKey), 'NUMBER') or
starts-with(normalize-space($inKey), 'STRING') ">
45      <xsl:value-of select="substring-before(normalize-space($inKey), '
')"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="'Unknown'"/>
50      </xsl:otherwise>
    </xsl:choose>
    </xsl:variable>
    <xsl:attribute name="name"><xsl:value-of
select="$varParamName"/></xsl:attribute>
55    <xsl:element name="SQTDataType"><xsl:attribute name="name"><xsl:value-of
select="$varParamDataType"/></xsl:attribute></xsl:element>
    <xsl:element name="SQLColumnRef">
      <xsl:call-template name="SQLColumnRef">
        <xsl:with-param name="inParamName" select="$varParamName"/>
60      </xsl:call-template>
    </xsl:element>
  </xsl:template>
    <!-- =====>
    <!-- SQLColumnRef                                -->
    <!-- =====>
65    <xsl:template name="SQLColumnRef">
      <xsl:param name="inParamName"/>

```

```

    <xsl:variable name="varTableName" select="substring-
before(key('keyColumnRef', concat('@', $inParamName))/@value, '.')"/>
    <xsl:variable name="varColumnName" select="substring-
after(key('keyColumnRef', concat('@', $inParamName))/@value, '.')"/>
5    <xsl:element name="SQLTable"><xsl:attribute name="name"><xsl:value-of
select="$varTableName"/></xsl:attribute></xsl:element>
    <xsl:element name="SQLColumn">
      <xsl:call-template name="SQLColumn">
        <xsl:with-param name="inColumnName" select="$varColumnName"/>
10      </xsl:call-template>
    </xsl:element>
  </xsl:template>
  <!-- =====>
  <!-- SQLColumn -->
15  <!-- =====>
  <xsl:template name="SQLColumn">
    <xsl:param name="inColumnName"/>
    <xsl:attribute name="name"><xsl:value-of
select="$inColumnName"/></xsl:attribute>
20    <xsl:element name="OleDbDataType">
      <xsl:attribute name="name"><xsl:value-of
select="'Unknown'"/></xsl:attribute>
      <xsl:attribute name="numericScale"><xsl:value-of
select="'none'"/></xsl:attribute>
25      <xsl:attribute name="numericPrecision"><xsl:value-of
select="'none'"/></xsl:attribute>
      <xsl:attribute name="dateTimePrecision"><xsl:value-of
select="'none'"/></xsl:attribute>
      <xsl:attribute name="characterLength"><xsl:value-of
30      select="'none'"/></xsl:attribute>
      <xsl:attribute name="isFixedLength"><xsl:value-of
select="'Unknown'"/></xsl:attribute>
      <xsl:attribute name="isLong"><xsl:value-of
select="'Unknown'"/></xsl:attribute>
35      <xsl:attribute name="scaleIsNegative"><xsl:value-of
select="'Unknown'"/></xsl:attribute>
    </xsl:element>
  </xsl:template>
  <!-- =====>
40  <!-- SQLStmtList -->
  <!-- =====>
  <xsl:template name="SQLStmtList">
    <xsl:for-each select="sections/section[@name='SQL']">
      <xsl:sort select="position()" data-type="number" order="ascending"/>
45      <xsl:element name="SQLStmt">
        <xsl:call-template name="SQLStmt">
          <xsl:with-param name="inNode" select="."/>
        </xsl:call-template>
      </xsl:element>
50    </xsl:for-each>
  </xsl:template>
  <!-- =====>
  <!-- SQLStmt -->
  <!-- =====>
55  <xsl:template name="SQLStmt">
    <xsl:param name="inNode"/>
    <xsl:variable name="varSQLText">
      <xsl:call-template name="concatSQLText">
        <xsl:with-param name="inElementList" select="item"/>
60      </xsl:call-template>
    </xsl:variable>
    <xsl:attribute name="text"><xsl:value-of
select="$varSQLText"/></xsl:attribute>
  </xsl:template>
65  <!-- =====>
  <!-- concatSQLText -->
  <!-- =====>
  <xsl:template name="concatSQLText">

```



```

    <xsl:param name="inElementList"/>
    <xsl:choose>
      <xsl:when test="$inElementList">
        <!--
5       The SQL will probably have equal signs in it, so concatenate key and
value attributes. Also
        preserve carriage returns & line feeds
        -->
        <xsl:variable name="varKey" select="normalize-
10 space($inElementList[position() = 1]/@key)"/>
        <xsl:variable name="varValue" select="normalize-
space($inElementList[position() = 1]/@value)"/>
        <xsl:variable name="varCurrentLine">
          <xsl:choose>
15            <xsl:when test="$varValue">
              <xsl:value-of select="concat($varKey, '=', $varValue)"/>
              <!-- Don't add newLine chars on the last input line -->
              <xsl:if test="$inElementList[position() !=
last()]"><xsl:text>&newLine;</xsl:text></xsl:if>
20            </xsl:when>
              <xsl:otherwise>
                <xsl:value-of select="$varKey"/>
                <!-- Don't add newLine chars on the last input line -->
                <xsl:if test="$inElementList[position() !=
25 last()]"><xsl:text>&newLine;</xsl:text></xsl:if>
              </xsl:otherwise>
            </xsl:choose>
          </xsl:variable>
          <xsl:variable name="varRemainingElements">
30            <xsl:call-template name="concatSQLText">
              <xsl:with-param name="inElementList"
select="$inElementList[position() != 1]"/>
            </xsl:call-template>
          </xsl:variable>
35          <xsl:value-of select="concat($varCurrentLine,
$varRemainingElements)"/>
          </xsl:when>
          <xsl:otherwise></xsl:otherwise>
        </xsl:choose>
40      </xsl:template>
      <!-- =====>
      <!--
      <!-- =====>
    </xsl:stylesheet>
45

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 2 U (http://www.xmlspy.com) by William
Barnum (Personal Copy) -->
50 <!-- *****-->
<!-- valSQTDef.xslt: validate SQTDef doc -->
<!-- *****-->

55 <!-- =====>
<!-- Internal general entities for boilerplate -->
<!-- =====>
<!DOCTYPE xsl:stylesheet [
60 <!ENTITY newLine "&#xd;&#xa;">
]>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:SQT2StoredProc="urn:SQT2StoredProc">
65 <xsl:output method="text" version="1.0" encoding="UTF-8" indent="no"/>
  <!-- =====>
  <!-- key lookups -->
  <!-- =====>

```

```

5      <!-- =====>
      <!-- global variables -->
      <!-- =====>
      <!-- root template -->
      <!-- =====>
      <!-- SQTDef -->
      <!-- =====>
10     <xsl:template match="SQTDef">
        <xsl:if test="string-length(@name)=0"><xsl:text>SQTDef element error:
missing name&newLine;</xsl:text></xsl:if>
        <xsl:apply-templates/>
        </xsl:template>
15     <!-- =====>
        <!-- SQTParameterList -->
        <!-- =====>
        <xsl:template match="SQTParameterList">
          <xsl:apply-templates/>
20        </xsl:template>
        <!-- =====>
        <!-- SQTParameter -->
        <!-- =====>
        <xsl:template match="SQTParameter">
25        <xsl:if test="string-length(@name)=0"><xsl:text>SQTParameter element
error: missing name&newLine;</xsl:text></xsl:if>
        -<xsl:if
test="SQT2StoredProc:paramNameIsValid(@name)=false()"><xsl:value-of
select="@name"/></xsl:if>
30        <!--<xsl:value-of select="SQT2StoredProc:paramNameIsValid('@name')"/> --
        >
        </xsl:template>
        <!-- =====>
        <!-- SQLColumnRef -->
        <!-- =====>
35        <xsl:template match="SQLColumnRef">
          <xsl:apply-templates/>
          </xsl:template>
        <!-- =====>
        <!-- SQLStmtList -->
        <!-- =====>
40        <xsl:template match="SQLStmtList">
          <xsl:apply-templates/>
          </xsl:template>
        <!-- =====>
        <!-- SQLStmt -->
        <!-- =====>
45        <xsl:template match="SQLStmt">
          <xsl:apply-templates/>
          </xsl:template>
50        <!-- =====>
        <!-- concatSQLText -->
        <!-- =====>
        <xsl:template match="parseSQLText">
55        </xsl:template>
        <!-- =====>
        <!-- =====>
        <!-- =====>
60    </xsl:stylesheet>

```

60 1.10.3 Sample INI file used with SQT2SP

[Startup]

SQTSourceFolder=E:\data\Source\vbs\SQT2SP\sqtFiles

StoredProcOutputFolder=E:\data\Source\vbs\SQT2SP\sqtFiles\

```

XsltFolder=E:\data\Visual Studio Projects\SQL2StoredProc\
SQTDef2StoredProcXsltFileName=SQTDef2SQLServerStoredProc.xslt
SQTDefValidatorXsltFileName=valsQTDef.xslt
DBConnectionString=Provider=SQLOLEDB;Data Source=localhost;Initial
5 Catalog=WDSS;Integrated Security=SSPI;
;DBConnectionString=Provider=MSDAORA.1;User
ID=myUID;password=myPWD;Data Source=myOracleServer;Persist Security
Info=False";

```

10 **1.11 'C' Source Code for ExpandTokenizedSQL**

A note on the fonts used in this section

Courier New 9pt Original source code
Times New Roman 12pt Comments added to explain the source.

15 **1.11.1 'C' Source Code for ExpandTokenizedSQL COM method**

THOMAS UPDATE HERE:

20 **1.11.2 'C' Source Code for rddExpandTokenizedSQL function**

The code is laid out with the subroutines defined before the main API routine.
This was done to avoid the need to forward reference the subroutine
25 prototypes.

```

/*-----
-----
    Tokenized SQL routines
-----*/
30 -----*/
#define MAX_TOKEN          32
#define MAX_TOKEN_LEN     16
#define TokenDelimiter    '&'
#define BeginToken        '@'
35
#define TokenString        1
#define TokenNumber        2
#define TokenDate          3
40 #define TokenUnknown     4

typedef struct    _Token
{
    char *    pszToken;
    int      nDatatype;
45    char *    pszValue;
}
Token,    *pToken;

```

_dbBuildTokenArrays parses the token/value pairs and builds a table of Token structures that will be used in the substitution pass of the SQL statement. The list of token/value pairs are a concatenation of the pszTokenValues parameter passed from the caller followed by the default token/values pairs found in the SQL source.

```

5      PRIVATE RC      _dbBuildTokenArrays(
        IN   DBC *    pDBC,
        IN   char *   pszTokens,
10      OUT  pToken    prgToken
      )
      {
        auto    int      iToken;
        auto    int      iDefault;
15      auto    int      iScan;
        auto    char *   pszTok;
        auto    char *   psz;
        static  char *   pszDatatype[] = {"STRING", "NUMBER", "DATE"};
        auto    int      iDatatype;
20      auto    char      chQuoteDB = chOption(OPT_QUOTE_CHAR);
        auto    char      chQuoteEX = chQuoteDB == '"' ?
        (char)'\'' : (char)'';
        auto    char      rgchQuote[3];
25      rgchQuote[0] = chQuoteDB;
        rgchQuote[1] = chQuoteEX;
        rgchQuote[2] = '\0';
30      /* Build an array of char * to the Key/Value pairs
        */
        MEMSET(prgToken, 0, sizeof(Token) * MAX_TOKEN);

        if (pszTokens[0] == '\0')
35      return RDD_SUCCESS;

        pszTok = pszTokens;
        iToken = 0;
        do
40      {
        /* See if the Token has a data type
        */
        for (iDatatype = 0; iDatatype < 3; ++iDatatype)
        {
45      if (dstrstri(pszTok, pszDatatype[iDatatype]) == pszTok)
        {
            while (*pszTok != ' ') pszTok++;
            while (isspace(*pszTok)) ++pszTok;
            break;
50      }
        }

        prgToken[iToken].nDatatype = iDatatype + 1;

```

```

5      /*      Get the KEY name
      */
      prgToken[iToken].pszToken = pszTok;
      while ((pszTok = strchr(pszTok, TokenDelimiter)) != NULL)
      {
10         /*      -----
            IF escaped delimiter THEN shift line left 1.
            Keep delimiter in the substitute value
            -----
      */

         if (pszTok[1] == TokenDelimiter)
         {
15             strcpy(pszTok, pszTok+1);
             pszTok++;
         }
         else
         {
20             *pszTok++ = '\0';
             break;
         }
      }

25      /*      Get the tokens substitution value
      */
      if ((psz = strchr(prgToken[iToken].pszToken, '=')) == NULL)
          return RDD_BAD_ARGUMENT_6;

30      *psz++ = NULL;
      alltrim(prgToken[iToken].pszToken);

      alltrim(psz);
      prgToken[iToken].pszValue = psz;

35      rtrim(prgToken[iToken].pszValue);
      }
      while (++iToken < MAX_TOKEN && pszTok);

```

```

/* -----
   Try to fix up those tokens passed or default which don't have a
   data type.
5   This takes two passes through the token array. The first pass
   will set
   any defaults not explicitly set in the SQT coding.
   -----*/
10  for (iScan = 0; iScan < 2; ++iScan)
    {
      for (iToken = 0; prgToken[iToken].pszToken != NULL; ++iToken)
        {
          if (prgToken[iToken].nDatatype == TokenUnknown)
15          {
            for (iDefault = iToken+1; prgToken[iDefault].pszToken;
            ++iDefault)
              {
                if (dstricmp(prgToken[iToken].pszToken,
20                prgToken[iDefault].pszToken) == 0)
                  {
                    prgToken[iToken].nDatatype =
prgToken[iDefault].nDatatype;
                    break;
25                  }
              }

            if (prgToken[iDefault].pszToken == NULL)
            {
30              if (prgToken[iToken].pszValue[0] == chQuotedB ||
                  prgToken[iToken].pszValue[0] == chQuoteEX
                  )
                prgToken[iToken].nDatatype = TokenString;
            else
35              prgToken[iToken].nDatatype = TokenNumber;
            }
          }

/* If there are quotes around a non-string value THEN remove
40 */
        if (prgToken[iToken].nDatatype != TokenString    &&
            (prgToken[iToken].pszValue[0] == chQuotedB    ||
             prgToken[iToken].pszValue[0] == chQuoteEX    )
        )
45        {
          *strchr(prgToken[iToken].pszValue + 1,
                  prgToken[iToken].pszValue[0]) = ' ';
          prgToken[iToken].pszValue[0] = ' ';
          alltrim(prgToken[iToken].pszValue);
50        }
      }
    }

55  return RDD_SUCCESS;
}

```

_dbExpandTokens

This is the work horse routine that takes the SQL source and token value routine finds and parses the [TOKENS] section of a tokenized SQL statement. A list of token/value pairs is built and passed back as the return value. If parameters were also passed into the API those token/value pairs are placed at the beginning of the concatenated list.

```

5  PRIVATE RC      _dbExpandTokens(
      IN   DBC *    pDBC,
      IN   char *   pszSQLSource,
10     OUT  char *   pszExpandedSQL,
      OUT  long *   pcbExpandedSQL,
      IN   char *   pszTokenValues
    )
    {
15     static char   szDelimiters[] = " ,=\\"'\\t\\n\\r()";
      auto char *   pszTokens;
      auto int      iToken;
      auto Token    rgToken[MAX_TOKEN];
      auto char     szToken[32];
20     auto int      cbToken;
      auto char *   pszSave;
      auto char *   psz;
      auto char *   pszIN;
      auto char *   pszOUT;
25     auto BOOL     bFoundToken;
      auto BOOL     bWork = FALSE;
      auto RC       rc      = RDD_SUCCESS;

      auto char     chQuoteDB = chOption(OPT_QUOTE_CHAR);
      auto char     chQuoteEX = chQuoteEX = chQuoteDB == "'" ?
        (char)'" : (char)'';

35     if ((pszTokens = (char *)ALLOC(STRLEN(pszTokenValues) + 16)) ==
        NULL)
        return RDD_NO_MEMORY;
        STRCPY(pszTokens, pszTokenValues);

40     rc = _dbBuildTokenArrays(pDBC, pszTokens, rgToken);
      if (rc != RDD_SUCCESS)
        goto _Exit_Here;

      if (pszExpandedSQL == NULL)
      {
45         bWork = TRUE;
        pszExpandedSQL = (char
          *)ALLOC(KiloBytes(iOption(OPT_SQL_WORK_BUFFER)));
      }
      pszOUT = pszExpandedSQL;
50     pszIN  = pszSQLSource;

      /* Check for correct quote characters and change to quotes used by
      DBMS */
55     _dbCheckQuotes(pDBC, pszSQLSource);

```

```

/* -----
-----
    All the work data is set up and the Token substitution array
built.
5
    Now go through the source SQL and copy to the Expanded buffer
inserting
    all token key values into the output stream.
-----
10
-----*/
    while (*pszIN)
    {
        if (*pszIN != BeginToken)
        {
15
            *pszOUT++ = *pszIN++;
            continue;
        }

20
        /* "Houston, we have a token" maybe
        */
        pszSave = pszIN++;          /* IF not valid we restore
        */
        cbToken = 0;

25
        for (psz = szToken; !strchr(szDelimiters, *psz); *psz++ =
*pszIN++)
        {
            ++cbToken;
            if (*pszIN == '\0' || cbToken == MAX_TOKEN_LEN)
30
                break;
        }
        *psz = '\0';          /* Terminate the copied token
        */

35
        if (cbToken == 0)          /* If an BeginToken alone
        */
        {
            *pszOUT++ = *pszSave++;    /* Copy the BeginToken
        */
            continue;
        }

40
        /* Find the token and substitute the value
        */
45
        bFoundToken = FALSE;
        for (iToken = 0; iToken < MAX_TOKEN; ++iToken)
        {
            if (rgToken[iToken].pszToken == NULL)
                break;

50
            if (dstrcmp(rgToken[iToken].pszToken, szToken) == 0)
            {
                psz = rgToken[iToken].pszValue;

55
            /* -----
            -----
                The value may have quote delimiters

                Check to make sure it is the correct quote char for the
                connected DBMS. Change if needed else just copy to SQL
            -----
            -----*/
60
            if (*psz == chQuoteEX)          /* IF the first char is a
            Quote */
            {
                *pszOUT++ = chQuoteDB;    /* Change to the correct
            Quote */
                psz++;

```



```
5      char      }
           else
           if (*psz == chQuoteDB)      /* IF the correct quote
           */                          *pszOUT++ = *psz++;      /* THEN just copy
           */
           else
           if (rgToken[iToken].nDatatype == TokenString)
               *pszOUT++ = chQuoteDB;
```

```

/* -----
5      Copy the value characters to the SQL
      Escape any embedded quotes by doubling the quote
character -----
-----*/
10      while (*psz)
        {
            if (psz[1] == '\\0')          /* IF on the last character
*/
                break;

15            if (*psz == chQuoteDB)      /* IF and embedded quote
*/
                *pszOUT++ = chQuoteDB; /* THEN escape by
doubling */
20            *pszOUT++ = *psz++;
        }

/* -----
25      The value may have quote delimiters
      Check the last character like the first (see above)
-----*/
30      if (*psz == chQuoteEX)          /* IF the first char is a
Quote */
        {
            *pszOUT++ = chQuoteDB;
            psz++;
35        }
        else
            if (*psz == '\\0')          /* IF the data is null
*/
            {
40                if (pszOUT[-1] == chQuoteDB)
                    {
                        --pszOUT;          /* Remove the opening quote
*/
                        ++pszIN;           /* Remove closing quote
45 */
                    }

                strcpy(pszOUT, "NULL");
                pszOUT += 4;
50                break;
            }
            else
55                *pszOUT++ = *psz++;          /* THEN just copy
*/

            if (rgToken[iToken].nDatatype == TokenString &&
                psz[-1] != chQuoteDB)
                *pszOUT++ = chQuoteDB;
60                break;
            }
        }

65      /* IF not a valid token with substitution then copy as is
      */
      if (iToken == MAX_TOKEN || rgToken[iToken].pszToken == NULL)
      {

```

```
        --pszIN;
        while (pszSave < pszIN)
            *pszOUT++ = *pszSave++;
5         }

        *pcbExpandedSQL = pszOUT - pszExpandedSQL;
        *pszOUT++ = '\0';

10     /* ----- */
        _Exit_Here:
        /* ----- */

        if (bWork)
15         FREE(pszExpandedSQL);

        FREE(pszTokens);

        return rc;
20     }
```

_dbGetDefaultTokens

This routine finds and parses the [TOKENS] section of a tokenized SQL statement. A list of token/value pairs is built and passed back as the return value. If parameters were also passed into the API those token/value pairs are placed at the beginning of the concatenated list.

```

5  PRIVATE  char *  _dbGetDefaultTokens(
      IN      char *  pszSourceSQL,
      IN      char *  pszTokenValues
10      )
    {
      auto    char *  psz;
      auto    char *  pszKeyWord;
      auto    char *  pszDefaults;
      auto    char *  pszAllTokens = NULL;
15     auto    char    szEndTokens[2] = {TokenDelimiter, 0};

        if ((pszKeyWord = dstrstri(pszSourceSQL, SECTION_TOKENS)) == NULL)
            return NULL;

20     for (psz = pszKeyWord; *psz != '['; --psz)
        *psz = '\0';
        *psz = '\0';

        while (*psz != ']') *psz++ = '\0';
25     *psz++ = '\0';

        while (isspace(*psz)) ++psz;

        pszDefaults = psz;
30     while (*psz)
        {
            if (*psz == '\n')
                *psz = TokenDelimiter;
            else
35             if (*psz == '\r')
                *psz = ' ';

            ++psz;
40         }

        psz = &pszDefaults[strlen(pszDefaults) - 1];
        while (*psz == TokenDelimiter || *psz == ' ')
            *psz-- = '\0';
45     pszAllTokens = (char *)ALLOC(strlen(pszTokenValues) +
        strlen(pszDefaults) + 32);

        if (strlen(pszTokenValues))
50         {
            strcpy(pszAllTokens, pszTokenValues);
            strcat(pszAllTokens, szEndTokens);
        }
        strcat(pszAllTokens, pszDefaults);
55     return pszAllTokens;
    }

```

_dbReadFileSQL

This routine reads a SQL statement from an external file. The filename can be a pathname or partial pathname.

```

5  PRIVATE RC      _dbReadFileSQL(char ** ppszSourceSQL)
   {
   auto    FILE *   pfSQL;
   auto    int      cb;

10      if ((pfSQL = fopen(*ppszSourceSQL, "rb")) == NULL)
          return RDD_FILE_NOT_FOUND;

   /* Get the size of the file
   */
15      fseek(pfSQL, 0, SEEK_END);
          cb = ftell(pfSQL);
          fseek(pfSQL, 0, SEEK_SET);

          if ((*ppszSourceSQL = (char *)ALLOC(cb + 64)) == NULL)
20          {
              fclose(pfSQL);
              return RDD_NO_MEMORY;
          }

25      /* Read the file into the allocated buffer
      */
          fread(*ppszSourceSQL, 1, cb, pfSQL);
          fclose(pfSQL);

30      return RDD_SUCCESS;
   }

```

_dbReadTableSQL

This routine reads a SQL statement from a table named SGSQl. The two columns that needs to be defined in the table are:

```

5      SQLName      contains the SQL name
      SQLText.      contains the SQL statement.
PRIVATE RC          _dbReadTableSQL(DBC * pDBC, char ** ppszSourceSQL)
{
10     auto    char    szQuery[256];
      auto    char    szTable[64];
      auto    char    szName[64];
      auto    char *   psz;
      auto    HTABLE   hTbl;
      auto    RC        rc;
15     auto    OptFlag  fOpt = {TRUE};
      auto    DBVALUE  dbValue;

      ValidatedDBC(pDBC);

20     psz = *ppszSourceSQL;
      if ((psz = strchr(psz, '.')) != NULL)
      {
          *psz++ = '\0';
          strcpy(szTable, *ppszSourceSQL);
25         strcpy(szName,  psz);
      }
      else
      {
          strcpy(szTable, "SGSQL");
30         strcpy(szName,  *ppszSourceSQL);
      }

      sprintf(szQuery,
35         "SELECT SQLName, SQLText FROM %s WHERE SQLName = '%s'",
          szTable, szName
      );

      rc = rddOpenQuery(pDBC, "GetSQLSource", szQuery, fOpt, &hTbl);
40     if (rc != RDD_SUCCESS)
        return rc;

      if ((rc = rddFetchRow(pDBC, hTbl, NULL, dbFirst)) == RDD_SUCCESS)
      {
          rc = rddGetData(pDBC, hTbl, "SQLText", NULL, &dbValue);
45         psz = (char *)ALLOC(dbValue.sColumnWidth + 32);

          if (psz == NULL)
              rc = RDD_NO_MEMORY;
50         else
            strcpy(psz, dbValue.pszChar);

            *ppszSourceSQL = psz;
      }
55     rddCloseHandle(pDBC, hTbl);

      return rc;
60 }

```

_dbFormatProcedureCall

If the call requested that a stored procedure be executed then this routine is called. The routine builds the EXECUTE statement adding the procedure name and its parameters and passes this EXECUTE statement in the expanded SQL buffer.

```

5 PRIVATE RC _dbFormatProcedureCall(
    IN DBC * pDBC,
    IN char * pszSQLSource,
    OUT char * pszExpandedSQL,
10 OUT long * pcbExpandedSQL,
    IN char * pszTokenValues
)
{
    auto Token rgToken[MAX_TOKEN]; /* Pointers to each KEY
    */
15 auto int iToken;
    auto char * pszWork;
    auto char * pszTokens;
    auto char * psz;
    auto int cb;
20 auto RC rc;

    UNREF(pDBC);

    pszTokens = (char *)ALLOC(STRLLEN(pszTokenValues) + 32);
25 if (pszTokens == NULL)
    return RDD_NO_MEMORY;

    STRCPY(pszTokens, pszTokenValues);

30 Build the token array.
    rc = _dbBuildTokenArrays(pDBC, pszTokens, rgToken);
    if (rc != RDD_SUCCESS)
    {
        FREE(pszTokens);
35 return rc;
    }

    juststem(pszSQLSource);

40 pszWork = (char *)ALLOC(strlen(pszSQLSource) +
    strlen(pszTokenValues) +
    (MAX_TOKEN * strlen(", ")));
    if (pszWork == NULL)
    {
        FREE(pszTokens);
45 return RDD_NO_MEMORY;
    }

    Begin building the EXECUTE SQL statement.
50 cb = sprintf(pszWork, "EXECUTE %s ", pszSQLSource);

    psz = &pszWork[cb];

```

Add each token/value pair as parameters to the EXECUTE statement.

```
for (iToken = 0; iToken < MAX_TOKEN; ++iToken)
{
    if (rgToken[iToken].pszToken == NULL)
5       break;

    psz += sprintf(psz, "@%s=%s, ", rgToken[iToken].pszToken,
10       rgToken[iToken].pszValue);
}
psz[-2] = '\\0';

*pcbExpandedSQL = strlen(pszWork);

if (pszExpandedSQL)
15     STRCPY(pszExpandedSQL, pszWork);

FREE(pszWork);
FREE(pszTokens);

20     return RDD_SUCCESS;
}
```


This is the public API called by user applications. This routine has three main sections:

1. Validate all input parameters
2. Based on the fSource parameter, get the SQL statement.
3. Process the SQL statement and apply tokens.

```

5
    /*-----
    -----
10    Name: rddExpandTokenizedSQL

        Expand a tokenized SQL statement and substitute the supplied
        values
    -----*/
15    RDDAPI(RC)      rddExpandTokenizedSQL(
        IN   DBC *    pDBC,
        IN   long     fSource,
        IN   char *    pszSourceSQL,
        OUT  char *    pszExpandedSQL,
20        OUT  long *   pcbExpandedSQL,
        IN   char *    pszTokenValues
    )
    {
25        auto    RC      rc;
        auto    int      fValidSource = fSource & 0x0F;
        auto    BOOL     bLoadedSQL   = FALSE;
        auto    char *    pszAllTokens = NULL;
        auto    char *    pszSQL;
        auto    char *    pszDFT;
30        auto    char *    pszCMT;

        Validate the input parameters.

35        if (pDBC)
            ValidateDBC(pDBC);

        if (pszSourceSQL == NULL)
            return RDD_BAD_ARGUMENT_3;

40        if (pszExpandedSQL != NULL)
        {
            if (!_bValidWriteMemory(pszExpandedSQL, *pcbExpandedSQL))
                return RDD_BAD_ARGUMENT_4;
        }

45        if (!_bValidWriteMemory(pcbExpandedSQL, sizeof(pcbExpandedSQL)))
            return RDD_BAD_ARGUMENT_5;

50        if (pszTokenValues == NULL)
            return RDD_BAD_ARGUMENT_6;
    
```

Get the SQL statement from one of the various sources. If the source is a stored procedure then the procedure is called directly and control returns to the caller. Each source type has its own read logic except when the SQL statement is passed in as a parameter (fValidSource == SQL_SOURCE_PARM)

```
5
/* Get the SQL source and Default token key-value pairs
*/
10  switch (fValidSource)
    {
        case SQL_SOURCE_FILE:
            if ((rc = _dbReadFileSQL(&pszSourceSQL)) != RDD_SUCCESS)
                return rc;

15         bLoadedSQL = TRUE;
        break;

        case SQL_SOURCE_TABLE:
            if ((rc = _dbReadTableSQL(pDBC, &pszSourceSQL)) !=
20  RDD_SUCCESS)
                return rc;

            bLoadedSQL = TRUE;
            break;

25         case SQL_SOURCE_PARM:
            break;

        case SQL_SOURCE_PROC:
30         return _dbFormatProcedureCall(pDBC, pszSourceSQL,
            pszExpandedSQL, pcbExpandedSQL, pszTokenValues
            );

35         default:
            return RDD_BAD_ARGUMENT_2;
    }
```

```

/* -----
We have everything we need. Now process the source.
5      1. Find and strip the SQL section header.
      2. Find and null all comment sections
      3. Find Default Tokens and concatenate with user passed tokens
      4. Expand the tokenized SQL
-----*/
10      if ((pszSQL = dstrstri(pszSourceSQL, SECTION_SQL)) != NULL)
          --pszSQL;

15      pszDFT = dstrstri(pszSourceSQL, SECTION_TOKENS);

Strip all the comment sections from the source.
      while ((pszCMT = dstrstri(pszSourceSQL, SECTION_COMMENTS)) !=
20      NULL)
      {
          if (pszCMT < pszSQL)
              MEMSET(pszCMT - 1, ' ', pszSQL - (pszCMT + 1));
          else
          if (pszCMT < pszDFT)
25              MEMSET(pszCMT - 1, ' ', pszDFT - (pszCMT + 1));
          else
              pszCMT[-1] = '\0';
      }

30      if (pszSQL != NULL)
          MEMSET(pszSourceSQL, ' ', (pszSQL+5) - pszSourceSQL);

Source files can have default tokens defined within the source. Build an array with
tokens passed in at call time in front of the default token definitions.
35      pszAllTokens = _dbGetDefaultTokens(pszSourceSQL, pszTokenValues);
      if (pszAllTokens)
          pszTokenValues = pszAllTokens;

Finally expand the SQL portion of the source substituting token values for every token
40      found.
      rc = _dbExpandTokens(pDBC, pszSourceSQL,
                          pszExpandedSQL, pcbExpandedSQL, pszTokenValues
                          );

45      If the source was a file or column in a table then a buffer was allocated to hold the
      source and needs to be freed before exiting.
      if (bLoadedSQL)
          FREE(pszSourceSQL);

50      if (pszAllTokens)
          FREE(pszAllTokens);

      return rc;
55  }

```

The foregoing description, for purposes of explanation, used specific nomenclature to provide a thorough understanding of the invention. However, it will be apparent to one skilled in the art that the specific details are not required in order to practice the invention. The foregoing descriptions of specific embodiments of the
 60 present invention are presented for purpose of illustration and description. They are

not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously many modifications and variations are possible in view of the above teachings. The embodiments are shown and described in order to best explain the principles of the invention and its practical applications, to thereby enable others
5 skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents: